2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

doi 10.26089/NumMet.v27r445

Dependency-to-constituency transformation

Pavel V. Grashchenkov
Lomonosov Moscow State University, Philological Faculty,
Department of Theoretical and Applied Linguistics, Moscow, Russia
Research Computing Center,
Laboratory of Automated Lexicographic Systems, Moscow, Russia
ORCID: 0000-0001-9754-2452, e-mail: graschenkovpv@my.msu.ru

Abstract: Dependency and constituency analyses are the two most widely used formalisms for representing natural-language syntax. Dependency grammar originates in traditional approaches to syntactic analysis and is now implemented in numerous computational systems. At the same time, theoretical linguists and linguistic typologists often favour constituency grammar because it is well suited to representing hierarchical phrase structure. Although the two formalisms have comparable descriptive power, dependency representations can be converted into constituency representations. This study proposes and implements an algorithm for dependency-to-constituency conversion. The algorithm identifies heads in a dependency representation and subsequently selects and linearizes the words belonging to the corresponding constituents. It is implemented as a Python library and has undergone initial testing.

Keywords: formal linguistics, dependency grammar, constituency grammar, syntax parsing, dependency tree, natural language syntax, natural language processing.

Acknowledgements: The study was conducted under the state assignment of Lomonosov Moscow State University.

For citation: P. V. Grashchenkov, “Dependency-to-constituency transformation,” Numerical Methods and Programming. **27** (4), 741–755 (2026). doi 10.26089/NumMet.v27r445.

© Pavel V. Grashchenkov

2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

Переход от грамматики зависимостей к грамматике составляющих

П. В. Гращенков

Московский государственный университет имени М. В. Ломоносова,
филологический факультет, кафедра теоретической и прикладной лингвистики,
Москва, Российская Федерация
Научно-исследовательский вычислительный центр,
лаборатория автоматизированных лексикографических систем,
Москва, Российская Федерация
ORCID: 0000-0001-9754-2452, e-mail: graschenkovpv@my.msu.ru

Аннотация: Анализ естественного языка в терминах зависимостей и составляющих представляет собой два наиболее распространенных способа формального описания синтаксиса. Грамматика зависимостей восходит к традиционным подходам к синтаксическому анализу и широко применяется в компьютерных системах обработки естественного языка. В то же время лингвисты-теоретики и лингвисты-типологи часто предпочитают описывать иерархическую синтаксическую структуру в терминах грамматики составляющих. Эти два формализма обладают сопоставимым описательным потенциалом, а представление в терминах зависимостей может быть преобразовано в представление в терминах составляющих. В настоящем исследовании предлагается и реализуется алгоритм преобразования зависимостного представления в представление в терминах составляющих. Алгоритм выявляет вершины в зависимостной структуре, а затем выбирает и упорядочивает слова, относящиеся к соответствующим составляющим. Алгоритм реализован в виде библиотеки на языке программирования Python и прошел начальное тестирование.

Ключевые слова: формальные модели в лингвистике, грамматика зависимостей, грамматика составляющих, синтаксический анализ, дерево зависимостей, синтаксис естественного языка, обработка естественного языка.

Благодарности: Исследование выполнено в рамках государственного задания Московского государственного университета имени М. В. Ломоносова.

Для цитирования: Гращенков П.В. Переход от грамматики зависимостей к грамматике составляющих // Вычислительные методы и программирование. 2026. **27**, № 4. 741–755. doi 10.26089/NumMet.v27r445.

---

1. Introduction. The formal study of syntactic representation emerged in the mid-twentieth century [1, 2], giving rise to two principal traditions: dependency grammar and constituency grammar. Dependency grammar originated in the work of L. Tesnière [3] and was further developed by I. A. Mel'čuk [4]. Its main formal properties are as follows: 1) a sentence is represented by head-dependent relations, forming a directed graph; 2) there is exactly one root word that has no head; 3) every other word has exactly one head; 4) at least one word is not a head; 5) the relation is not symmetric, i.e. if h is the head of d, then d cannot be the head of h; 6) the relation is acyclic, i.e. if h is the head of d and b is the head of h, then d cannot be the head of b.

More formally, let S be the set of words in a sentence and let H be the set of heads. The dependency representation is defined as follows:

$$\forall h \in H: \ \exists i \in S, \ h = \mathrm{head}(i),$$

$$\forall d \in S, \ \forall h \in H: \ h = \mathrm{head}(d) \Rightarrow \neg \exists j \in S: \ \big(j \neq h \wedge j = \mathrm{head}(d)\big),$$

$$\exists d \in S: \ d \notin H,$$

$$\forall d \in S, \ \forall h \in H: \ h = \mathrm{head}(d) \Rightarrow \neg\, d = \mathrm{head}(h),$$

$$\forall d \in S, \ \forall h, b \in H: \ h = \mathrm{head}(d) \ \& \ b = \mathrm{head}(h) \Rightarrow \neg\, d = \mathrm{head}(b).$$

2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

Figure 1 shows a dependency representation of the English sentence “I saw a UFO in the backyard”.

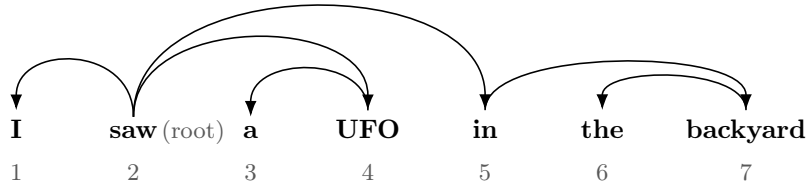


Figure 1. Dependency tree for the sentence “I saw a UFO in the backyard”

Constituency grammar is most closely associated with N. Chomsky, who proposed a detailed phrase-structure analysis of English [5] and developed it further in later works [6, 7]. The most recent Chomskyan framework, called Minimalism, received a formal mathematical treatment in [8]. In constituency grammar, sentence structure is represented by relations of constituent inclusion: words belong to constituents, which may themselves be contained in larger constituents. Its core properties are as follows: 1) every word belongs to at least one constituent; 2) one constituent contains all other constituents; 3) a constituent may be contained in another constituent; 4) constituent inclusion is properly nested, which ensures that no constituent extends beyond the boundaries of its host; 5) there are no two constituents with exactly the same boundaries.

More formally, let S be the set of words in a sentence and let C be the set of constituents. The constituency representation is defined as follows:

$$\forall i \in S: \ \exists cP \in C, \ i \in cP,$$
$$\exists CP: \ \forall cP \in C, \ cP \subsetneq CP,$$
$$\forall cP \in C \setminus \{CP\}: \ \exists aP \in C, \ cP \subsetneq aP,$$
$$\forall aP, bP, cP \in C: \ cP \subsetneq aP, \ cP \subsetneq bP \Rightarrow aP \subsetneq bP \text{ or } bP \subsetneq aP,$$
$$\forall aP, cP \in C: \ aP \subsetneq cP \Rightarrow \neg\, cP \subsetneq aP.$$

In the representation used here, each constituent has a head, which determines its type, e.g. NP for noun phrase and VP for verb phrase. Figure 2 shows a constituency representation of the English sentence “I saw a UFO in the backyard”.

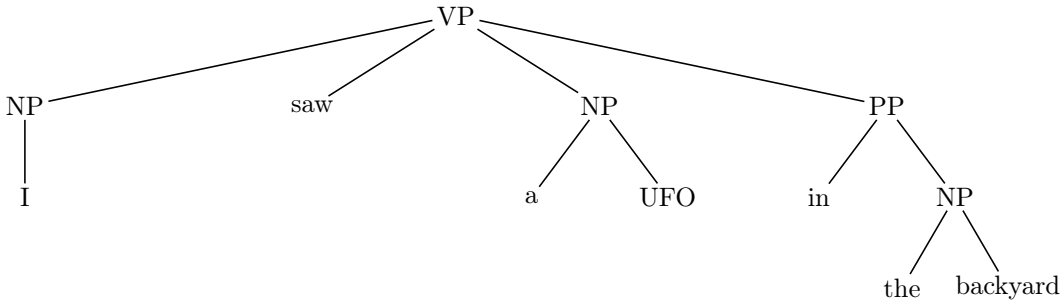


Figure 2. Constituency tree for the sentence “I saw a UFO in the backyard”

The same constituent structure can be expressed by bracketed annotation:

```
[VP [NP I] saw [NP a UFO] [PP in [NP the backyard]]].
```

The grammar hierarchy established in [9, 10] distinguishes four classes of grammars (in increasing order of expressive power): regular, context-free, context-sensitive, and unrestricted. Under the assumptions adopted here, dependency and constituency grammar are weakly equivalent to context-free grammars [11]. Both representations are widely used in computational linguistics [2]. Dependency grammar has often been implemented in computational systems, for example, in the Link Grammar project [12], whereas constituency-based representations remain central to much linguistic theorizing.

2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

The practical motivation for conversion is interoperability between these traditions. Large UD (Universal Dependencies) corpora and dependency parsers provide readily available dependency analyses, whereas many linguistic queries, corpus-search procedures, and downstream analyses are formulated over phrase spans. The intended output of `dep2con` is therefore a compact, head-based bracketing that makes contiguous phrase domains available for such uses; it is not intended to reconstruct a gold Penn Treebank or a full GB/Minimalist analysis. This restricted goal explains why a simplified constituent representation is useful even though it does not retain every distinction encoded in a richly annotated constituency tree.

2. Descriptive potential of dependency and constituency grammar. Although dependency and constituency representations can often encode comparable syntactic information, constituency-based analyses are frequently preferred when explicit phrase structure is needed. They make hierarchical grouping, phrase boundaries, movement, and phrasal scope directly visible in the representation. This does not imply that dependency grammar cannot model these phenomena; rather, it may require additional representational machinery such as enriched relations, features, or empty nodes. Several relevant cases are discussed below.

2.1. Hierarchical ordering of dependents. Within a noun phrase, dependent types often occur in a constrained hierarchical order. For example:

```
[QP all [NumP three [AdjP big [NP ring roads]]]].
```

The order “all three big ring roads” is acceptable, whereas permutations such as “*big all three ring roads” are ungrammatical (the asterisk marks ungrammaticality). A dependency representation can attach all modifiers to the lexical head roads, but it does not by itself encode the fact that the quantifier “all” scopes over the constituent “three big ring roads”. A constituency representation captures this naturally through nested phrase structure. A comparable issue arises in wh-questions:

```
[CP Who_i [C did] [TP John [VP meet t_i]]].
```

The wh-phrase originates as the object of “meet” but is realized in the left periphery of the clause. Constituency structure represents both its displacement and the clausal domain in which it occurs. A dependency representation can account for this relation using enhanced dependencies or empty nodes, but a basic dependency tree does not directly represent the moved phrase and its gap in the same way.

2.2. Modification of phrases instead of individual words. Some modifiers are semantically related to a phrase or construction rather than to a single lexical head. For example:

```
The ceiling was [DegP too [AP low]] [PP for me].
```

The phrase “for me” expresses the standard relative to which the degree phrase “too low” is evaluated. A flat dependency representation has to choose a lexical attachment point, typically “low” or “too”, even though the interpretation concerns the larger degree construction. Constituency structure can represent this grouping explicitly.

2.3. Different syntactic behavior of words and phrases. The distribution of a lexical adjective may differ from that of an adjectival phrase. For instance, let us compare two sentences:

“We need a trained specialist” and “We need a specialist trained specifically for us”.

A single adjective is readily placed before the noun, whereas a modified adjectival phrase is normally placed after it. Conversely, “*a specifically trained for us specialist” is not acceptable. Constituency structure directly distinguishes the lexical item “trained” from the larger adjectival phrase “trained specifically for us”, allowing grammatical rules to refer to their different structural positions.

2.4. Phrase-internal adjacency effects in phonological processes. Some phonological alternations depend on linear adjacency rather than direct syntactic dependency. For example, English uses the article “an” before a vowel sound: “an ongoing process”; “*a ongoing process”.

In a dependency analysis, both the determiner and the adjective may depend on the noun process, so their phonological interaction is not expressed by the dependency relation itself. A constituency analysis identifies both words as members of the same noun phrase, while linear order specifies their adjacency. This example supports the utility of constituent domains at the syntax-phonology interface.

2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

2.5. Scope-taking expressions. Expressions such as negation and focus particles often take semantic scope over phrases. For example, "not qualified doctors" and "only qualified doctors". The interpretation of "not" or "only" depends on the constituent over which it takes scope. In these examples, negation or restriction may apply either to "qualified" or to "doctors": "They are not qualified doctors, but qualified teachers" / "They are not qualified doctors, they are untrained"; "Only qualified doctors can be invited to this position, not teachers" / "Only qualified doctors can be invited to this position, not untrained ones". A constituency representation makes potential scope domains explicit, such as an NP or a larger clause. In a dependency representation, the particle must attach to a particular head word, although its semantic scope may extend beyond that word.

2.6. Insufficiency of one head for a dependent morphological feature assignment. Let us consider the following example of a phrase in Russian: "Собака кажется больной" ("The dog seems ill").

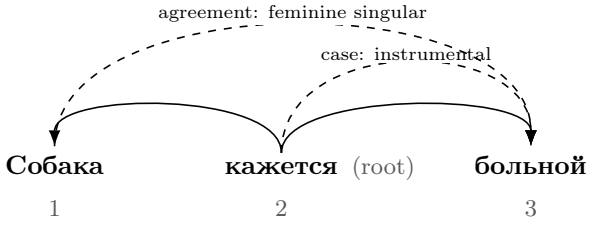


Figure 3. Dependency relations and feature control in the sentence "Собака кажется больной"

In a basic dependency tree, the word "кажется" is the syntactic head of both "Собака" and "больной". However, two distinct relations determine the form of the predicative adjective "больной". Its instrumental case is selected by the predicate "кажется", whereas its feminine singular form is controlled by agreement with the subject "Собака". The two controllers are indicated in Figure 3.

This example shows a restriction of a pure one-head dependency representation: a tree of unlabeled word-to-word links cannot simultaneously express the predicate's case selection and the subject's agreement control. One of these relations must be encoded separately, for example, as a feature constraint, a secondary dependency, or an enhanced graph edge. The constituency formalism allows different feature requirements to be satisfied within their respective phrasal domains. Thus, just as continuous constituency representations require enrichment to handle non-projective structures, a pure dependency tree requires enrichment to represent certain cases of co-dependency at the level of morphosyntactic features.

2.7. Ellipsis and anaphora. Ellipsis often involves the omission of material that corresponds to a phrase. For example, "John ate a banana, and Mary did too". The auxiliary "did" stands for the omitted VP "ate a banana". Similarly, in gapping constructions such as "John ate a banana, and Mary an apple", interpretation requires recovering parallel clause-level structure. Constituency representations provide natural units: NP, VP, or clause for describing the antecedent and the omitted material. Dependency-based analyses can model ellipsis, but they commonly need more apparatus: empty nodes, enhanced graphs, or reconstruction procedures to recover the relevant phrase-like domain. Anaphora resolution also often benefits from constituent information, because potential antecedents may be nominal phrases or larger discourse units rather than isolated words.

2.8. Abstraction and grammatical generalization. Constituency formalisms facilitate generalizations over phrases. For instance, a grammar may state that a verb selects a complement realized as an NP, PP, or clause. Likewise, an English preposition may combine with an NP regardless of whether its head is a proper noun, common noun, or pronoun: "for John", "for this person" or "for me". A constituency grammar can express this with the following schematic rule: PP $\rightarrow$ P NP. This generalization abstracts away from the lexical category of the NP head. Dependency frameworks can express comparable generalizations through dependency labels, POS constraints, and feature structures.

In summary, constituency representations are particularly useful when an analysis must refer to phrase boundaries, hierarchical containment, movement, ellipsis, anaphora, or scope domains. Their advantage is primarily descriptive and explanatory: they make these structural units explicit, whereas dependency representations focus primarily on relations between individual words.

3. Previous research. Modern computational approaches to syntactic analysis matured around the turn of the millennium [13, 14] and now achieve high accuracy through neural methods. One example is Stanza, the Python library used in the present `dep2con` project. Stanza is a PyTorch-based neural NLP pipeline [15]. Given

2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

raw text, it performs sentence segmentation and tokenization, lemmatization, part-of-speech and morphological tagging, named-entity recognition, and syntactic parsing. Its native syntactic representation is UD: for each word, the parser predicts a syntactic head and a labelled dependency relation, producing a dependency tree. For the present task, which disregards dependency labels, the relevant accuracy of head prediction is approximately 90% for Russian and 89% for English.

The conversion of dependency annotation into constituency annotation has been investigated previously. Three lexicalized probabilistic parsing models were developed in [13] to improve constituent-parsing accuracy and to recover richer predicate–argument information. The models were trained and evaluated on the Wall Street Journal portion of the Penn Treebank using labelled precision, labelled recall, and crossing-bracket measures. The first model is a probabilistic lexicalized context-free grammar that generates each constituent by selecting a head and then its left and right modifiers. The second model distinguishes complements from adjuncts and introduces probabilistic subcategorization frames. The third model additionally represents wh-movement and traces, thereby supporting the recovery of long-distance dependencies. The reported performance reaches 93.3% labelled precision and 90.1% labelled recall.

It was shown in [16] that dependency trees can be converted into phrase structures with high fidelity, although the information available in dependency annotation is insufficient for a one-to-one reconstruction of Chomskyan GB/Minimalist trees. Three conversion algorithms were proposed in that work. The first assumes that every category has two projection levels, an intermediate X' and a maximal XP [6]. Although this representation is common in theoretical linguistics, it produces many nodes and is not straightforward to adapt to applied systems. The second is a flat conversion scheme, which was proposed in [14]. Each head has a single projection, and a dependent projects only when it has dependents of its own. This approach is compact, but it loses distinctions among several phrase types. The third algorithm generalizes the first two and uses three configurable, treebank-specific resources: 1) a projection table specifying valid projection chains for each category; 2) an argument table specifying the arguments a head may take and their positions; 3) a modification table specifying modifiers, their positions, and their attachment types.

More recent works illustrate two different goals for dependency-to-constituency conversion. One approach [17] encodes constituent structure as head-ordered dependency trees and learns the required ordering information, including for discontinuous structures. Another [18] formulates conversion as structured prediction and learns phrase-structure output from annotated training data. The third [19] converts SynTagRus into an enriched Penn Treebank-style representation using language- and treebank-specific rules; this procedure explicitly resolves non-projectivity before the conversion. The Russian-language algorithm in [20] likewise constructs a constituency tree from a dependency tree. In contrast, `dep2con` is a deterministic, format-oriented tool: it consumes UD/CoNLL-U input, does not require a constituency training corpus or language-specific projection tables, and produces simplified bracketed constituents rather than a treebank-specific, information-preserving phrase-structure tree.

Most current tools for morphosyntactic annotation use the UD/CoNLL-U format [21]. UD is a cross-linguistic framework for the consistent annotation of morphology and dependency syntax. It provides a universal inventory of part-of-speech tags, morphological features, and dependency relations, while allowing documented language-specific subtypes where needed. CoNLL-U is the tab-separated plain-text format used to serialize UD annotations. Each sentence is represented by comments, normally including `sent_id` and `text`, one row for each syntactic word and a blank line marking the sentence boundary. Each word row has exactly ten fields. The basic `HEAD`+`DEPREL` representation forms a rooted dependency tree: every word except the root has one head, whereas the root has `HEAD=0` and `DEPREL=root`.

The UD project maintains treebanks for many languages at https://universaldependencies.org/. New languages and data are added regularly. Consequently, large quantities of automatically parsed and manually annotated linguistic data are available and may provide a basis for deriving constituency representations across languages.

At the same time, relatively few software tools address dependency-to-constituency conversion. The archived `wenkokke/dep2con` project (https://github.com/wenkokke/dep2con) is implemented in Haskell. It receives an explicitly nested dependency tree with word, part-of-speech, and serial-number fields and recursively applies a conversion procedure similar to that described in [14]. In contrast, the present library reads ordinary CoNLL-U rows or raw text parsed with Stanza, performs UD-specific normalization, and returns bracketed phrase spans. The difference is therefore primarily one of input representation, integration, and output scope, rather than a claim that the two implementations solve unrelated problems.

2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

4. Target representation and algorithm specification. The aim of the project is to develop a general dependency-to-constituency conversion method that relies only on the existence and direction of links between words. Accordingly, the method does not use dependency-relation labels. This choice supports generality across annotation schemes and has a practical motivation: modern neural dependency parsers, including Stanza, may predict the head of a word correctly while misclassifying the relation label.

The target representation is not a standard GB/Minimalist tree, but a simplified flat, non-binary structure similar, though not identical, to the "naive word-based grammar" of [22]. It does not encode projection levels such as SBAR or S; in this respect, it is closer to the flat conversion scheme discussed in [16] and [14]. The representation also does not distinguish specifiers, complements, and adjuncts, because this information is difficult to recover reliably from an unlabeled dependency structure [6, 23].

The present construction is defined for projective dependency trees. For a non-projective tree, the linear intervals associated with subtrees can overlap without proper nesting (Figure 4); a single bracketed phrase-structure representation cannot express such crossings. The current version therefore does not claim to produce a well-formed nested constituent tree for non-projective input. Such input requires a separate projectivisation or a representation supporting discontinuous constituents, as in papers [19, 17].

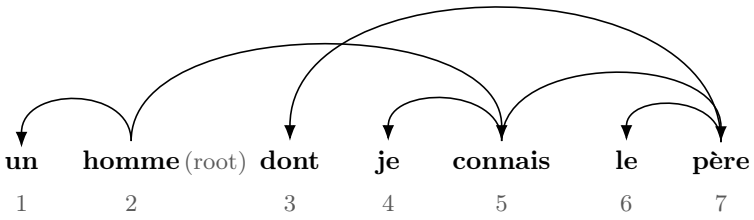


Figure 4. Non-projective dependency tree for the French phrase "un homme dont je connais le père"

As the French example shows, strong non-projectivity is attested in natural languages. In dependency representations, however, such structures can be encoded as configurations of mutually crossing head-dependent links. Constituency representations encounter difficulties because the relevant constituents are not properly nested: one word may belong to two overlapping constituents that do not contain one another. Contemporary generative grammar addresses this problem by introducing an additional structural level of constituents. Unfortunately, this solution is not readily available in a computational implementation. Consequently, `dep2con` may assign an incorrect label in such cases.

Dependency-relation labels are deliberately not used in the core construction. This choice increases portability across UD-like data and isolates the contribution of dependency direction and word order, but it also prevents the system from recovering distinctions such as complement versus adjunct or a full inventory of functional projections. Labels are retained in the internal representation and may be used in a future, language-specific extension; their omission is a scope limitation, not a claim that they are linguistically unimportant.

The algorithm begins by identifying all heads in a sentence, where S is the set of words. The corresponding pseudocode is given in Listing 1 in Appendix 1.

Next, for each head i, the algorithm identifies the constituent iP headed by i. This is a top-down procedure, from heads to dependents; a bottom-up procedure would also be possible in principle (Listing 2 in Appendix 1).

Constituent construction also requires an ordering of elements. If a dependency representation provides no ordering information, the construction of an ordered constituent is underdetermined. In CoNLL-U, however, token identifiers form a monotonically increasing sequence and therefore provide the required linear order (Listing 3 in Appendix 1).

This procedure identifies constituents from head-dependent relations and orders their members by token identifiers. The largest constituent corresponds to the sentence. Additional normalization is required because UD conventionally annotates adpositions and copulas as dependents in some constructions. When necessary, the algorithm restructures relations in adpositional phrases and copular predications before constructing constituents.

The complete procedure is summarized below:

1. Input data: sentence in dependency format.

2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

2. Preprocessing:

   a) preposition rearrangement;

   b) copula processing.

3. Main processing:

   a) head identification — finding words that have dependents;

   b) child search — determining dependents for each head;

   c) correctness check — validating the dependency structure;

   d) bracket determination — defining positions for opening and closing brackets.

4. Output formation: final postprocessing — formatting the final result, including insertion of brackets into the sentence.

The complete workflow is shown in Figure 5 in Appendix 1.

5. Implementation details. The project is implemented in Python and is entirely rule-based. It does not use machine-learning methods for the conversion itself.

The module converts a sentence-level dependency representation into a hierarchical constituent structure. It takes a dependency tree, in which words are connected by head-dependent relations, and constructs constituents headed by nouns, verbs, adjectives, and other lexical categories. The implementation uses an iterative procedure with recursive validation in the `check_head` function. It collects head-dependent relations and the word forms associated with each head. The main function, `make_const`, accepts either raw text or the path to a CoNLL-U file, processes each sentence, and returns a structured representation.

The workflow consists of five main stages, which are described below.

5.1. Parsing input data. The module operates in two modes:

1. It uses Stanza (`use_parser=True`) to perform tokenization, morphological analysis, and dependency parsing on raw text.

2. It loads a pre-existing CoNLL-U file (`use_parser=False`), which is useful for processing previously annotated corpora.

5.2. Internal sentence representation. Each sentence is converted into an internal `SentenceData` dictionary. Its keys are word identifiers; its values contain the word form, lemma, part of speech, head identifier, and dependency-relation type.

5.3. Structure normalization and validation. This stage resolves differences between UD conventions and the constituent-oriented representation used by the module. UD dependency trees do not always align with traditional assumptions about constituency. The module therefore performs two normalization operations:

1. `rearrange_pp` promotes an adposition (ADP) in the tree. In UD, a preposition usually depends on a noun. For the present constituent representation, the preposition governs the nominal complement, as in `[PP [IN in] [NP backyards]]`. The function reverses the relevant arc so that the noun becomes dependent on the preposition.

2. `rearrange_cop` processes copular verbs such as “be” and “seem”. It makes the copula the root of the sentence and attaches the nominal predicate, e.g. “doctor” in the sentence “He is a doctor”, as its dependent. It also reassigns selected dependents, including subjects and adverbial modifiers, to the copula.

The `check_head` function then validates the dependency structure, primarily by detecting cycles. For each node, the function follows the chain of heads and reports an error if the chain returns to a previously visited node. This validation is particularly important for dependency structures produced by neural parsers. The function also checks that the sentence has a valid root node.

2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

5.4. Bracketed structure construction. After the dependency tree has been normalized, the module extracts constituents. The `heads` function returns the identifiers of words that govern at least one dependent. The `find_children` function then collects the dependents of each head and stores their identifiers and word forms.

The `find_brackets_l` and `find_brackets_r` functions determine the left and right boundaries of each constituent and place the corresponding brackets. For every head and its dependents, the algorithm finds the leftmost and rightmost member of the constituent. If a dependent occurs to the left or right of the head, the appropriate bracket is placed at the outermost element on that side; otherwise, it is placed at the head. Terminal words in the `h_terminals` list may be enclosed in a constituent according to user-defined settings. By default, single-word nominals and adverbs (`NOUN`, `PRON`, `PROPN`, and `ADV`) are enclosed in constituents.

5.5. Finalization and output. The `post_proc` function adjusts bracket positions relative to punctuation. The `put_brackets_sent` function produces the final bracketed representation by inserting brackets labelled with phrase types, e.g. `[NP ...]` and `[VP ...]`. Phrase type is determined from the part of speech of the head using the `PHRASE_DICT` dictionary. The `select_spec_cons` function extracts selected constituent types, such as noun phrases, for subsequent analysis.

Thus, the tool provides a bridge between two major paradigms of syntactic analysis and allows their respective advantages to be combined.

6. Library usage and evaluation. The `dep2con` library is published on PyPI (https://pypi.org/project/dep2con/), the public package repository for Python. It can be installed with `python -m pip install dep2con`. The main function accepts a single required argument: either a text string or CoNLL-U input. When the input is a text string, it is first processed by Stanza; the language and the `use_parser` parameter must then also be specified. Listings 4 and 5 in Appendix 1 show raw-text input and iteration over the results.

The output includes the bracketed parse of the full sentence and the constituents extracted from it. Constituent types and other output parameters are configurable. The following example shows the sentence-level parse and extracted constituents:

```
[VP [NP I] saw [NP a UFO] [PP in [NP the backyard]]].

NPs:
I
a UFO
the backyard.

VPs:
I saw a UFO in the backyard.

PPs:
in the backyard.
```

The input parameters control the parsing language, orthographic treatment, terminal categories to be represented as phrases, and the constituent types selected for extraction. For example, the `emb` parameter returns all embedded phrases when set to `True`, which is the default:

```
NPs:
I
a UFO of my friend
my friend
my
the backyard of my professor
my professor
my.

VPs:
I saw a UFO of my friend in the backyard of my professor.
```

2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

```
PPs:
of my friend
in the backyard of my professor
of my professor.
```

When `emb=False`, only maximal projections are returned:

```
NPs:
I
a UFO of my friend
the backyard of my professor.

VPs:
I saw a UFO of my friend in the backyard of my professor.

PPs:
of my friend
in the backyard of my professor.
```

The `dep2con.sent_dict` attribute provides the UD morphosyntactic representation of a sentence as a Python dictionary; an abbreviated example is given in Listing 6 in Appendix 1. The CoNLL-U input mode is used in the same way (Listing 7 in Appendix 1).

The corresponding output is as follows:

```
[NP l'année [PP de [NP mon bac]]].

NPs:
l'année de mon bac
mon bac.

PPs:
de mon bac
...
```

The program also provides additional output formats.

Robustness testing was conducted on the complete French ParisStories corpus (1 387 sentences) [24], TOROT UD (26 496 sentences) [25], and SynTagRus UD (61 889 sentences) [26]. Each run returned one `dep2con` result per input sentence; no runtime failures or skipped sentences were observed. This result establishes robustness of input processing, rather than linguistic correctness of every resulting constituent structure.

The Penn Treebank experiment used the complete fixed set of 390 records (IDs 1–390) in the NLTK Treebank extraction [27]. Every record was processed by Stanza version 1.9.2 and converted by `dep2con`; no sentence was excluded after parsing. Evaluation on these sentences assessed constituent boundaries without considering phrase type. It yielded a precision of 74.9%, a recall of 52.6%, and an F_1-score of 61.8%.

We also evaluated the conversion on Turkish data derived from the UD and constituency source treebanks of Parallel Trees [28]. Because the public source repositories are not globally aligned, we retained only sentence pairs with exactly matching token sequences, a unique constituency counterpart, and a projective dependency tree. After removing 68 one-token items, the evaluation set comprised 348 multiword UD-constituency pairs. Morphological and other non-syntactic metadata, punctuation-only constituents, and phrase-category labels were removed before comparison; one unlabeled terminal span was then added around every remaining token in both representations. Under this category-agnostic, terminal-normalized evaluation, `dep2con` obtained 95.61% precision, 90.79% recall, and 93.14% F_1-score. This score evaluates agreement of the complete normalized bracketed representations; it is not directly comparable with an evaluation restricted to non-singleton phrase boundaries.

Finally, we conducted an English consistency evaluation on 100 sentences of 7–10 words selected from public-domain literary works available through Project Gutenberg [29]. Each sentence was processed once with Stanza to produce both a UD-style dependency analysis and a native constituency analysis; `dep2con` was

2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

applied to the former. The Stanza output was normalized to the `dep2con` notation by removing non-substantive projection and punctuation wrappers. Labelled constituent spans were then compared, while punctuation and one-token spans were ignored. All 100 pairs had identical non-punctuation token sequences. The conversion achieved 68.14% precision, 82.01% recall, and 74.43% F_1 against the normalized Stanza analyses. Since both representations were produced by Stanza, this experiment is a consistency comparison rather than an evaluation against independently manually annotated gold constituents.

For each evaluation, an exact match was a predicted constituent span with the same token boundaries as a gold constituent span after the preprocessing described above. Let M be the number of matching spans, P the number of predicted spans, and G the number of gold spans. Precision, recall, and the F_1-score were calculated as:

$$\text{Precision} = \frac{M}{P}, \qquad \text{Recall} = \frac{M}{G}, \qquad F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Table 1 summarizes the evaluation results.

Table 1. Summary of evaluation results

Evaluation	Pairs	Labels	Precision, %	Recall, %	F_1-score, %
Penn Treebank	390	ignored	74.90	52.60	61.80
Parallel Trees (Turkish)	348	ignored	95.61	90.79	93.14
Stanza consistency (English)	100	retained	68.14	82.01	74.43

Some errors may be attributable to incorrect dependency parses produced by Stanza. For example, in the first Penn Treebank sentence, the sequence “a nonexecutive director Nov. 29” is analyzed as an NP, incorrectly including “Nov. 29”. Errors also arise from mismatches in phrase boundary considerations. For instance, Stanza’s constituency parser produces the representation `[NP [NP cricket] [NP match]]`, whereas `dep2con` produces `[NP [NP cricket] match]`. The intended structure is the same; the difference lies in whether the projecting constituent head is enclosed in brackets.

7. Conclusion. We implemented `dep2con`, a Python library that converts dependency trees into simplified bracketed constituency representations. The library accepts raw text through the Stanza pipeline as well as pre-existing CoNLL-U data. It allows users to include or exclude embedded phrases and supports the extraction of selected constituent types. The approach is intentionally based on dependency direction and word order rather than dependency-relation labels, which makes it applicable to a broad range of UD-style annotations. Future work will include a more comprehensive evaluation of constituent precision, recall, and F_1-score across different treebanks.

References

1. Ya. G. Testelets, *Introduction to General Syntax* (Russian State University for the Humanities, Moscow, 2001) [in Russian].
2. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*, 3rd ed. Online manuscript (2026). https://web.stanford.edu/~jurafsky/slp3/. Cited November 15, 2026.
3. L. Tesnière, *Éléments de syntaxe structurale* (Klincksieck, Paris, 1959).
4. I. A. Mel’čuk, *Dependency Syntax: Theory and Practice* (State University of New York Press, Albany, NY, 1988).
5. N. Chomsky, *Syntactic Structures* (Mouton, The Hague, 1957).
6. N. Chomsky, *Lectures on Government and Binding* (Foris Publications, Dordrecht, 1981).
7. N. Chomsky, “Minimalist Inquiries: The Framework,” in *Step by Step: Essays on Minimalist Syntax in Honor of Howard Lasnik* (MIT Press, Cambridge, MA, 2000), pp. 89–155.
8. E. P. Stabler, “Derivational Minimalism,” in *Logical Aspects of Computational Linguistics: First International Conference, LACL’96, Nancy, France, September 23–25, 1996*, Lecture Notes in Computer Science, Vol. 1328. (Springer, Berlin, 1997), pp. 68–95. doi 10.1007/BFb0052152.
9. N. Chomsky, “Three Models for the Description of Language,” IRE Trans. Inf. Theory **2** (3), 113–124 (1956). doi 10.1109/TIT.1956.1056813.

2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

10. N. Chomsky, "On Certain Formal Properties of Grammars," Inf. Control **2** (2), 137–167 (1959). doi 10.1016/S0019-9958(59)90362-6.
11. H. Gaifman, "Dependency Systems and Phrase-Structure Systems," Inf. Control **8** (3), 304–337 (1965). doi 10.1016/S0019-9958(65)90232-9.
12. D. D. K. Sleator and D. Temperley, "Parsing English with a Link Grammar," in *Proc. Third International Workshop on Parsing Technologies, Tilburg, Netherlands and Durbuy, Belgium, August 10–13, 1993* (Association for Computational Linguistics, 1993), pp. 277–292.
13. M. Collins, "Three Generative, Lexicalised Models for Statistical Parsing," in *Proc. 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics* (Madrid, Spain, 1997), pp. 16–23. doi 10.3115/976909.979620.
14. M. Collins, J. Hajič, L. Ramshaw, and C. Tillmann, "A Statistical Parser for Czech," in *Proc. 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics, College Park, Maryland, USA, 1999* (Association for Computational Linguistics, 1999), pp. 505–512. doi 10.3115/1034678.1034754.
15. P. Qi, Y. Zhang, Y. Zhang, et al., "Stanza: A Python Natural Language Processing Toolkit for Many Human Languages," in *Proc. 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2020, pp. 101–108. doi 10.18653/v1/2020.acl-demos.14.
16. F. Xia and M. Palmer, "Converting Dependency Structures to Phrase Structures," in *Proc. First International Conference on Human Language Technology Research, San Diego, March 18–21, 2001*, (Association for Computational Linguistics, Stroudsburg, USA, 2001), pp. 1–5. https://doi.org/10.3115/1072133.1072147. Cited November 15, 2026.
17. D. Fernández-González and A. F. T. Martins, "Parsing as Reduction," in *Proc. 53rd Annual Meeting of the Association for Computational Linguistics and 7th International Joint Conference on Natural Language Processing, Beijing, China, July 26–31, 2015*, Vol. 1. (Association for Computational Linguistics, 2015), pp. 1523–1533. doi 10.3115/v1/P15-1147.
18. L. Kong, A. M. Rush, and N. A. Smith, "Transforming Dependencies into Phrase Structures," *2015 Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, May 31–June 5, 2015.* In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics, 2015*, pp. 788–798. doi 10.3115/v1/N15-1080.
19. A. Luu, S. A. Malamud, and N. Xue, "Converting SynTagRus Dependency Treebank into Penn Treebank Style," in *Proc. 10th Linguistic Annotation Workshop held in conjunction with ACL 2016 (LAW-X 2016), Berlin, Germany, August 11, 2016* (Association for Computational Linguistics, 2016), pp. 16–21. doi 10.18653/v1/W16-1703.
20. A. Yu. Poletaev, I. V. Paramonov, and E. I. Boychuk, "Algorithm of Constituency Tree from Dependency Tree Construction for a Russian-Language Sentence," Informatics and Automation **22** (6), 1323–1353 (2023). https://doi.org/10.15622/ia.22.6.3. Cited November 15, 2026.
21. M.-C. de Marneffe, C. D. Manning, J. Nivre, and D. Zeman, "Universal Dependencies," Comput. Linguist. **47** (2), 255–308 (2021). doi 10.1162/coli_a_00402.
22. M. Ermolaeva, "Learning Syntax via Decomposition," Ph.D. thesis (University of Chicago, Chicago, IL, 2021). doi 10.6082/uchicago.3015.
23. N. Chomsky, *The Minimalist Program* (MIT Press, Cambridge, MA, London, 1995).
24. S. Kahane, B. Caron, E. Strickland, and K. Gerdes, "Annotation Guidelines of UD and SUD Treebanks for Spoken Corpora: A Proposal," in *Proc. 20th International Workshop on Treebanks and Linguistic Theories (TLT, SyntaxFest 2021), Sofia, Bulgaria, 2021* (Association for Computational Linguistics, 2021), pp. 35–47. doi 10.18653/v1/2021.tlt-1.4.
25. H. M. Eckhoff and A. Berdicevskis, "Linguistics vs. Digital Editions: The Tromsø Old Russian and OCS Treebank," Scripta & e-Scripta **14–15**, 9–25 (2015).
26. Ye. S. Inshakova, L. L. Iomdin, L. G. Mityushin, et al., "The SynTagRus Today," Proceedings of the V. V. Vinogradov Russian Language Institute **3** (21), 14–41 (2019). [in Russian]. doi 10.31912/pvrli-2019.3.1.
27. S. Bird, E. Loper, and E. Klein, *Natural Language Processing with Python: analyzing text with the natural language toolkit* (O'Reilly Media, Inc., Sebastopol, CA, 2009).
28. C. Alzetta, A. Miaschi, F. Dell'Orletta, et al., "Parallel Trees: A Novel Resource with Aligned Dependency and Constituency Syntactic Representations," Lang. Resour. Eval. **59** (4), 3445–3485 (2025). doi 10.1007/s10579-025-09826-3.
29. Free eBooks | Project Gutenberg. https://www.gutenberg.org/. Cited November 15, 2026.

2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

Received	*Accepted*	*Published*
August 22, 2026	*October 21, 2026*	*November 20, 2026*

Information about the author

Pavel V. Grashchenkov — PhD. Habil. in Philology, Associate Professor; Head of the Laboratory; 1) Lomonosov Moscow State University, Philological Faculty, Department of Theoretical and Applied Linguistics, Leninskie Gory, 1, 119234, Moscow, Russia; 2) Research Computing Center, Laboratory of Automated Lexicographic Systems, Leninskie Gory, 1 build.4, 119991, Moscow, Russia.

2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

Appendix 1
Supporting algorithm listings and workflow

This appendix contains the detailed pseudocode, usage examples, and workflow diagram referred to in the main text.

Listing 1. Head-identification pseudocode

```
for s in S:
        if s.head() == i:
                H.add(i)
```

Listing 2. Child-search pseudocode

```
for i in H:
        for s in S:
                if s.head() == i:
                        iP.add(s)
```

Listing 3. Constituent-ordering pseudocode

```
iP_ordered = iP.sorted()
```

Listing 4. Raw-text input

```
output = make_const("I saw a UFO in the backyard", lang = 'en', use_parser = True)
```

Listing 5. Iteration over the conversion results

```
for dep2con in output:
    print(dep2con.sent_parse)
    for i in dep2con.x_phrases:
        print(i)
```

Listing 6. Abbreviated `sent_dict` representation

```
{
  1: {'idw': 1, 'wf': 'I', 'nf': 'I', 'pos': 'PRON',
      'tags': 'Case=Nom|Number=Sing|Person=1|PronType=Prs',
      'head_id': 2, 'rel': 'nsubj'},
  2: {'idw': 2, 'wf': 'saw', 'nf': 'see', 'pos': 'VERB',
      'tags': 'Mood=Ind|Number=Sing|Person=1|Tense=Past',
      'head_id': 0, 'rel': 'root'},
  ...
  7: {'idw': 7, 'wf': 'backyard', 'nf': 'backyard', 'pos': 'NOUN',
      'tags': 'Number=Sing', 'head_id': 5, 'rel': 'case'}
}
```

Listing 7. CoNLL-U input

```
output = make_const("fr_parisstories-ud-train.conllu")
```

2026, **27** (4), 741–755. doi 10.26089/NumMet.v27r445

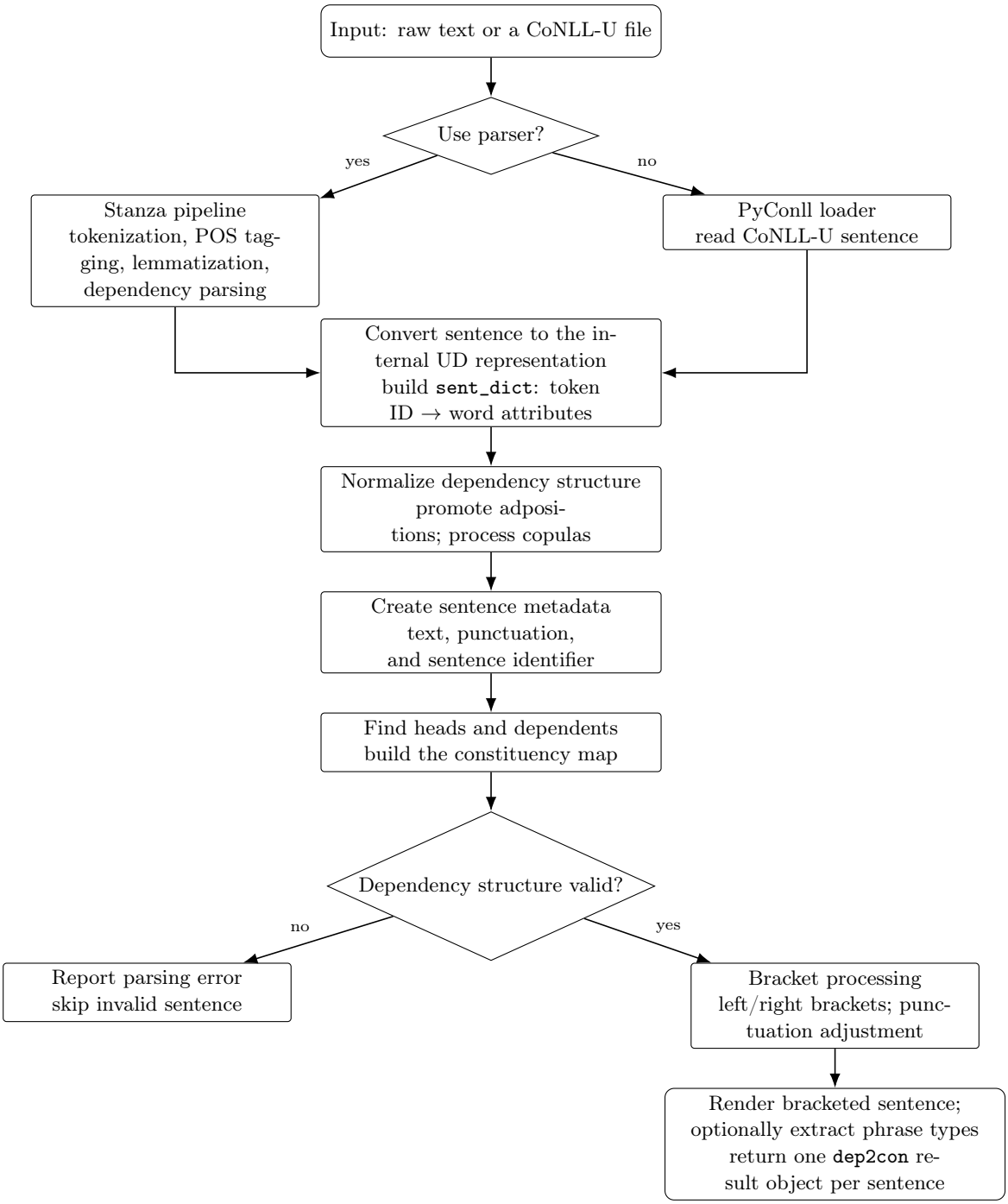


Figure 5. Flowchart of the dependency-to-constituency conversion procedure